IDK1531 Course Project

Your task is to create file encoder/decoder using Huffman statistical compression algorithm.

Huffman coding is a lossless data compression algorithm, which assigns variable-length prefix-free codes to input characters such that more frequent characters get shorter codes, and the less frequent characters get larger codes.

1 Input and Output

The program compress and decompress files, and accepts the following command line options:

–i specifies the input file

–o specifies the output file

Read file inputFile as input, compress it and write the compressed version to outputFile.

```
mysolution -i inputFile -o outputFile
```

The same can be accomplished by using only the –o modifier as follows:

```
mysolution -o outputFile inputFile
```

The solution must be able to work with character streams and work with stdin/stdout streams.

```
mysolution < inputFile > outputFile
```

The program must return 0 upon successful completion and a nonzero exit code otherwise.

The output file contains a header and the compressed input file. The header is a compact representation of a Huffman codebook, allowing to reconstruct the codebook from this representation using canonical Huffman codes.

2 Huffman Codebook

Suppose we have an input file containing 33 bytes:

```
0xA 0xB 0xA 0xC 0xA 0xD 0xC 0xD 0xA 0xA 0xD 0xA 0xA 0xA 0xC
0xA 0xA 0xC 0xC 0xD 0xC 0xD 0xD 0xC 0xD 0xC 0xC 0xC 0xC 0xC
0xC 0xC 0xC
```

2.1 Compact Representation of the Huffman Codebook

The first step in the encoding process is to create a Huffman codebook, corresponding to a given input. This is accomplished in a series of 3 steps.

2.1.1 Calculate the frequency map

First, you need to calculate the frequencies of every byte in the input file. In the case of our example, the frequency map is the following.

Byte	0xA	0xB	0xD	0xC
Frequency	10	1	7	15

2.1.2 Construct the Huffman Tree

The next step involves constructing a Huffman tree from the frequency map. A Huffman tree is a binary tree, in which the frequency of the left child is less than the frequency of the right child.

A Huffman tree may be constructed using a heap structure, which sorts tree nodes by their frequency. In the beginning, for every byte in an input file, construct a corresponding tree node, containing the byte value and its frequency. These will be leaf nodes in your tree. Push all the leaf nodes into the heap.

Table 1: Step 1.

Node	Frequency
0xB	1
0xD	7
0xA	10
0xC	15

Until there is only one node left in the heap, proceed as follows:

1. Pop two elements from the heap (the ones with smallest frequencies)
2. Construct a new tree node with the two elements you just popped from the heap as its children, with the node having less frequency being the left-hand child. The frequency of this tree node is the sum of the frequencies of its children.
3. Push the newly created tree node into the heap.
4. Repeat until there is just one tree node left in the heap.

(a) Step 2.

Node	Frequency
(0xB,0xD)	8
0xA	10
0xC	15

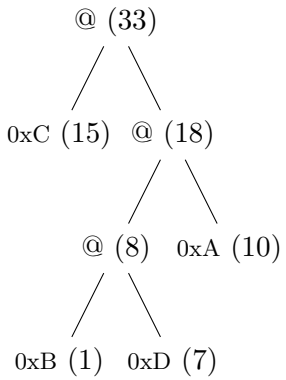
(b) Step 3.

Node	Frequency
((0xB,0xD),0xA)	18
0xC	15

(c) Step 4.

Node	Frequency
(0xC,((0xB,0xD),0xA))	33

Once the process is finished, there is just one element in the heap, which is the root node of the tree



2.1.3 Group symbols into bins by code length

The code length of a specific symbol is the level at which the symbol resides in a Huffman tree. The level of the root node is 0. Sort entries by code length, and sort the symbols, sharing the same code length, by value.

Code length	Symbols
1	0xC
2	0xA
3	0xB, 0xD

2.1.4 Huffman header

As a last step, construct the Huffman header, which is the compact representation of the Huffman codebook. A Huffman header consists of two parts: a) the number of symbols with each code length – the length table, and b) symbols themselves – the symbols' table.

Given the table of symbols grouped by code lengths into bins as shown in example of the previous task, the Huffman header looks as follows:

```
1, 1, 2 :: 0xC, 0xA, 0xB, 0xD
```

It means that:

- there is one code of length 1 for symbol 0xC.
- there is one code of length 2 for symbol 0xA.
- there are two codes of length 3 for symbols 0xB and 0xD

Note that the comas and the :: symbol are for decorative purposes only, and have no meaning in general.

The Huffman header is the compact representation of the Huffman codebook, and is written into the compressed file, followed by the compressed contents of the file.

2.2 Canonical Huffman Codebook

The canonical Huffman codebook is constructed from the Huffman header as follows.

1. Start on row 0 of the length table with first_code_on_row=0 and first_index_on_row=0 and write them out.
2. The next value for first_code_on_row = first_code_on_row + number_of_codes shifted by 1 to the left.
3. The next value for first_index_on_row = first_index_on_row + number_of_codes.
4. If there are more rows in the length table, go to step 2 and repeat.

The Huffman codebook corresponding to the header

```
1, 1, 2 :: 0xC, 0xA, 0xB, 0xD
```

is the following:

Table 3: Huffman Codebook

First code on row	First index on row	Number of codes	Symbols
0	0	1	0xC
10	1	1	0xA
110	2	2	0xB, 0xD

This table means that symbol 0xC is encoded as a single bit 0, symbol 0xA is encoded as two bits 10, symbol 0xB is encoded as 110, and 0xD is encoded as 111. The codes are consecutive within a single row.

3 Encoding

The encoding process is simply transforming every symbol into its binary representation in accordance with the codebook.

Assume we wish to encode a sequence of bytes

```
0xA 0xB 0xC 0xD
```

using a codebook from Table 3.

The bitstring corresponding to the encoded sequence is 101100111, and the contents of the compressed file are

```
[Huffman header] 101100111
```

4 Decoding

Assume we have a compressed file

```
[Huffman header] 101100111
```

The decoding process is done in two steps. First, we reconstruct the Huffman codebook from the Huffman header, and then decode the payload according to the codebook.

4.1 Reconstructing the Huffman codebook

Suppose we have reconstructed the Huffman header

```
1, 1, 2 :: 0xC, 0xA, 0xB, 0xD
```

from the compressed file, and from the Huffman header we have reconstructed the Huffman codebook following the procedures described above.

First code on row	First index on row	Number of codes	Symbols
0	0	1	0xC
10	1	1	0xA
110	2	2	0xB, 0xD

4.2 Decoding

To decode the compressed file, proceed as follows.

1. Start on row 0 of the length table with first_code_on_row=0 and first_index_on_row=0 and write them out.
2. Fetch one bit from the bitstream and write it into fetched_bits.
3. If fethed_bits − first_code_on_row < number_of_codes, then the index of the symbol index is fethed_bits − first_code_on_row + first_index_on_row. Output the symbol.
4. Add number_of_codes to first_code_on_row shifted by 1 to the left.
5. Add number_of_codes to first_index_on_row.
6. Subtract first_code_on_row from fetched_bits and first_code_on_row.
7. Go to step 2 for the next row of the length table.

Suppose we want to decompress payload 0x01 0x66, which in binary notation corresponds to a bitstring

```
101100110
```

We start fetching bits from the bitstring.

Fetched bits	First code on row	Fetched - First	First index on row	Number of codes	Action
1	0	1	0	1	Continue to the next row
10	10	0	1	1	Index = 0+1 = 1. Output 0xA.
1	0	1	0	1	Continue to the next row.
11	10	1	1	1	Continue to the next row
110	110	0	2	2	Index = 0+2 = 2. Output 0x0B.
0	0	0	0	1	Index = 0+0 = 0. Output 0xC.
1	0	1	0	1	Continue to the next row.
11	10	1	1	1	Continue to the next row
111	110	1	2	2	Index = 1+2 = 3. Output 0x0D.

Write the decompressed sequence 0xA 0xB 0xC 0xD to the output file.